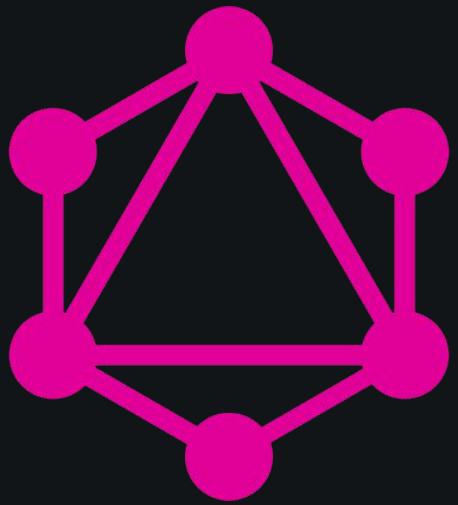
NODE.JS NIGHTS GRAPHQL

Josef Zavisek

STRV

AGENDA

- Introduction
- GraphQL language
- GraphQL server
- Demo



GraphQL

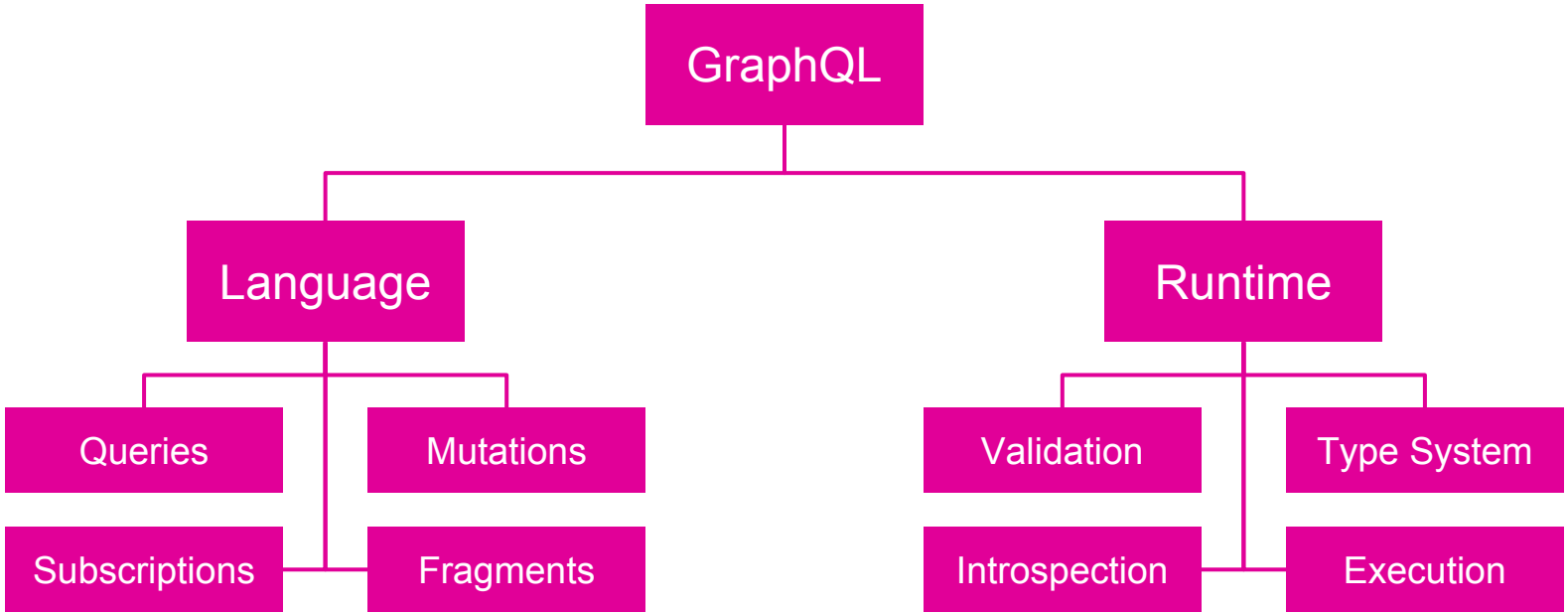
STRV

WHAT IS NOT GRAPHQL

- ✕ Database
- ✕ Query language for particular database technology
- ✕ One specific implementation (apollo, express-graphql, graphql-ruby …)

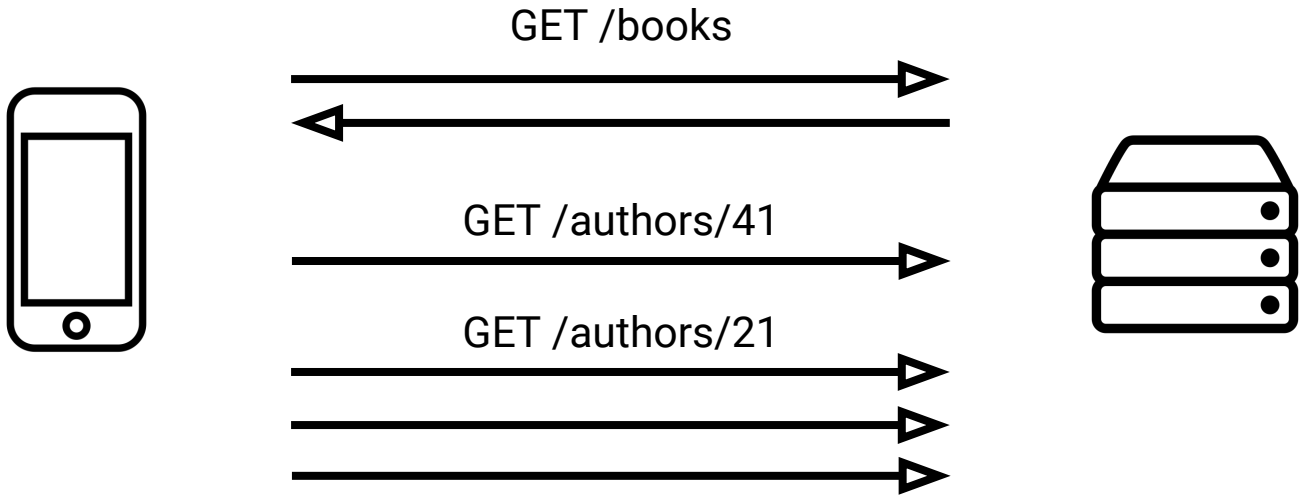
WHAT IS GRAPHQL

- Query language for APIs & server-side runtime for executing queries



WHAT'S WRONG WITH REST API?

- Too many round-trips to the server



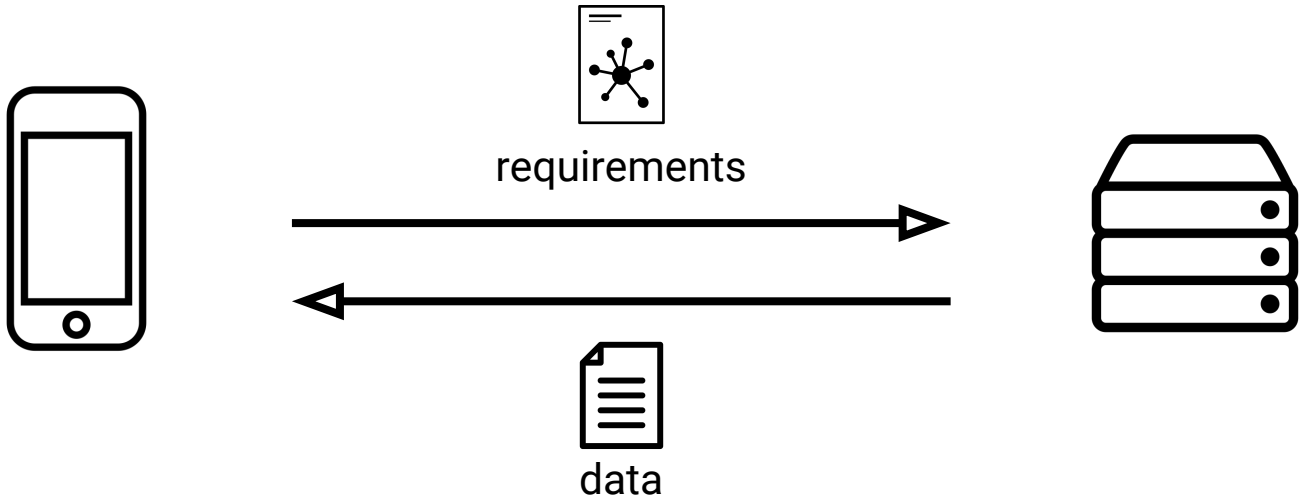
- GET /books?include=authors&authorFields=id,name&bookFields=name,gender

EXPENSIVE FIELDS

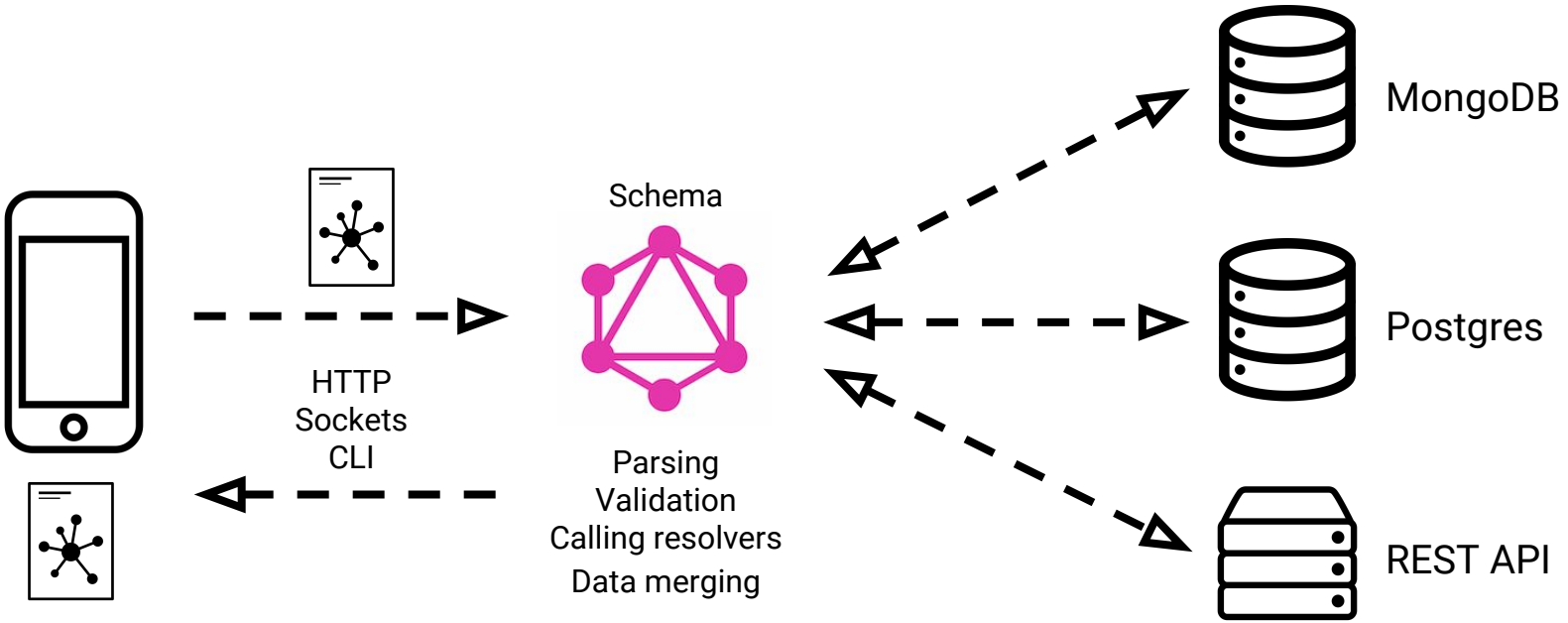
- Some fields might be hard to compute
- Overfetching
- Unpredictable shape of data

GRAPHQL SOLUTION

- Client specifies all the requirements at once



HOW DOES IT WORK?



GRAPHQL LANGUAGE

GRAPHQL LANGUAGE

```graphql
query {
  dogs {
    id
    name
    breed
    birthYear
    photo
  }
}
```

```json
{
  "data": {
    "dogs": [
      {
        "id": 1,
        "name": "Belle",
        "breed": "beagle",
        "birthYear": 2018,
        "photo": "https://images.dog.ceo/n02088364_10731.jpg"
      },
      {
        "id": 2,
        "name": "Dollie",
        "breed": "terrier",
        "birthYear": 2006,
        "photo": "https://images.dog.ceo/n02093754_3442.jpg"
      }
    ]
  }
}
```

GRAPHQL LANGUAGE - QUERIES

- Name is optional
- Nested objects allowed

```
query getDogs {
  dogs {
    id
    name
    breed
    birthYear
    photo
    user {
      id
      name
    }
  }
}
```

GRAPHQL LANGUAGE - QUERY VARIABLES

```graphql
query getDogs($breed: String!) {
  dogs(filter: { breed: $breed }) {
    id
    name
    breed
    birthYear
    photo
    user {
      id
      name
    }
  }
}
```

```json
{
  "breed": "husky"
}
```

GRAPHQL LANGUAGE - ALIASES

```
query getDogs {
  belle: dog(id:1) {
    id
    name
    breed
    birthYear
    photo
  }
  dollie: dog(id: 2) {
    id
    name
    breed
    birthYear
    photo
  }
}
```

```
{
  "data": {
    "belle": {
      "id": 1,
      "name": "Belle",
      "breed": "beagle",
      "birthYear": 2018,
      "photo": "https://images.dog.ceo/n02088364_10731.jpg"
    },
    "dollie": {
      "id": 2,
      "name": "Dollie",
      "breed": "terrier",
      "birthYear": 2006,
      "photo": "https://images.dog.ceo/n02093754_3442.jpg"
    }
  }
}
```

GRAPHQL LANGUAGE - FRAGMENTS

```graphql
query getDogs {
  belle: dog(id:1) {
    ... DogFields
  }
  dollie: dog(id: 2) {
    ... DogFields
  }
}

fragment DogFields on Dog {
  id
  name
  breed
  birthYear
  photo
}
```

```json
{
  "data": {
    "belle": {
      "id": 1,
      "name": "Belle",
      "breed": "beagle",
      "birthYear": 2018,
      "photo": "https://images.dog.ceo/n02088364_10731.jpg"
    },
    "dollie": {
      "id": 2,
      "name": "Dollie",
      "breed": "terrier",
      "birthYear": 2006,
      "photo": "https://images.dog.ceo/n02093754_3442.jpg"
    }
  }
}
```

GRAPHQL LANGUAGE - DIRECTIVES

```graphql
query getDogs($withUser: Boolean!) {
  dogs {
    id
    name
    breed
    birthYear
    photo
    user @include (if: $withUser) {
      id
      name
    }
  }
}
```

```json
{
  "withUser": true
}
```

GRAPHQL LANGUAGE - MUTATIONS

```graphql
mutation createDog($input: CreateDogInput!) {
  createDog(input: $input) {
    id
    name
    breed
    birthYear
    photo
  }
}
```

```json
{
  "input": {
    "name": "Dollie",
    "breed": "terrier",
    "birthYear": 2018
  }
}
```

GRAPHQL SERVER

SERVER

- We will use “apollo-server-koa” package

```
// Create Apollo server
const server = new ApolloServer({
  typeDefs,
  resolvers,
  debug: true,
  introspection: true,
  context: makeContext,
  playground: playgroundConfig,
  engine: engineConfig,
  formatError,
})

// Apply Apollo middleware
server.applyMiddleware({
  app,
  path: '/graphql',
})
```

SCHEMA / TYPEDEFS

- Usually in .gql or .graphql files
- We can use “merge-graphql-schemas” package to load it

```
type User {
  id: Int!
  name: String!
}

type Dog {
  id: Int!
  name: String!
  breed: String!
  birthYear: Int!
  photo: String
  age: Int!
  user: User
}

type Query {
  dog(id: Int!): Dog
  dogs: [Dog]
}
```

RESOLVERS

- Their structure has to match the schema

```
'use strict'

const operations = require('../../operations/dogs')

module.exports = {
  Query: {
    dog: (root, args) => operations.getById({ id: args.id }),
    dogs: () => operations.getAll(),
  },
  Dog: {
    // Computed field
    age: dog => dog.age || new Date().getFullYear() - dog.birthYear,
  },
}
```

LET'S CODE

QUESTIONS

THAT'S IT

Josef Zavisek / @jzavisek, @strvcom

STRV

STRV